Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

TPRVLA:Temporal-Preference-Recovery Vision-Language-Action Model

Qiyun Sun
Northeast Forestry University
Lin Shi
Northeast Forestry University
Zihao Chu
Tongji University
Yiyu Han
Northeast Forestry University
Meiling Liu
mlliu@nefu.edu.cn

Northeast Forestry University

Article

Keywords: VLA, Temporal Memory, Direct Preference Optimization, Anomaly Detection, Closed Loop Recovery

Posted Date: November 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11497368/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

TPRVLA:Temporal-Preference-Recovery Vision-Language-Action Model

Qiyun Sun[1], Lin Shi[2], Zihao Chu[3], Yiyu Han[1], Meiling Liu[2*]

[1]Aulin college, Northeast Forestry University, Harbin, China.
[2]School of computer science and artificial intelligence, Northeast Forestry University, Harbin, China.
[3]Shanghai Research Institute for Intelligent Autonomous Systems, Tongji University, Shanghai, 200092, China.

*Corresponding author(s). E-mail(s): mlliu@nefu.edu.cn;
Contributing authors: sqyzyyx@nefu.edu.cn; 1378524246@nefu.edu.cn; zihaoai@outlook.com; hanyiyu_060914@nefu.edu.cn;

Abstract

Vision-language-action (VLA) models offer a promising approach to robotic manipulation, yet reliable long-horizon execution in dynamic environments remains challenging. OpenVLA supports strong task generalization and parameter-efficient adaptation, but its single-image input limits temporal modeling, while its standard framework lacks explicit use of failure experience and runtime recovery mechanisms. To address these limitations, we propose TPRVLA, which integrates temporal memory, preference learning, and closed-loop recovery. A lightweight temporal memory adapter incorporates consecutive observations into the frozen OpenVLA backbone to capture short-term dynamics. Weighted preference pairs are constructed from successful and failed trajectories under similar visual observations and identical instructions. The policy is then adapted through a joint direct preference optimization (DPO) and behavior cloning (BC) objective to favor preferred actions. During execution, a rule-based state monitor combines end-effector motion, action repetition, and visual changes to trigger checkpoint-guided recovery when anomalies persist. Evaluations across 900 rollouts on ten LIBERO Spatial tasks with three random seeds show improved task performance, particularly on tasks requiring extended execution and closed-loop adjustment, while increasing peak GPU memory usage by less than 1%.

Keywords: VLA, Temporal Memory, Direct Preference Optimization, Anomaly Detection, Closed Loop Recovery

1 Introduction

With the rapid development of large pretrained models in visual information understanding and natural language reasoning, the method of transferring related knowledge to the perception, decision making and execution of embodied agents has become a popular and crucial direction in robot learning. Early works such as SayCan and PaLM-E introduced language models and multimodal models for task planning, environmental understanding and reasoning[1][2], then RT-1 and RT-2 unified visual observations, natural language instructions, and robot actions, advancing robot policies from imitation learning for individual tasks toward generalist policy learning driven by large and heterogeneous trajectory datasets[3][4], Open X-Embodiment further aggregate the operation data from multiple robot platforms, laying the data foundation of knowledge transfer across tasks and embodiments[5]. These advances established a scalable learning paradigm that combines pretrained multimodal representations with heterogeneous robot trajectories, laying the foundation for knowledge transfer and capability generalization in generalist robot policies.. Under this background, developing VLA models that can understand natural language instructions, exploit complex visual information and transfer across manipulation tasks has become an important pathway toward moving embodied intelligence from laboratory evaluation to practical applications.

However, real robot manipulation involves continuous dynamics, differences across embodiments and environmental disturbances. Datasets and benchmarks such as LIBERO and DROID further reflect the diversity of manipulation scenes, tasks, and robot platforms[6][7]. Some of learned robot policies rely mainly on successful expert demonstrations and learn mappings from current observations to control commands through action regression, action chunking, or probability distribution modeling[8][9], making it difficult to explicitly exploit the relative preferences between successful and failed trajectories. During long running task execution, small perception or control errors can gradually accumulate which may lead to a final false. Previous studies on runtime monitoring and environmental feedback reasoning have shown that timely use of execution feedback, progress assessment and corrective actions is essential for improving the reliability of robot manipulation[10][11]. Therefore, exploiting observation history, failure experience and immediate execution feedback jointly under limited training and constrained computational resources remains a critical challenge for VLA models.

In the relevant research, OpenVLA has become an important foundation model for open-source VLA exploration[12]. Built upon the pretrained backbone of Prismatic VLMs[13], OpenVLA combines the spatial visual features of DINOv2 with the semantic visual features of SigLIP and uses large-scale demonstrations across tasks and robot embodiments from Open X-Embodiment[14][15][5]. In addition to releasing its model weights, training code and data processing pipeline, OpenVLA demonstrates that the adaptation of consumer-grade GPUs can be performed through LoRA and model quantization[16]. This provides a unified and extensible foundation of reproducing and improving for generalist robot policies.

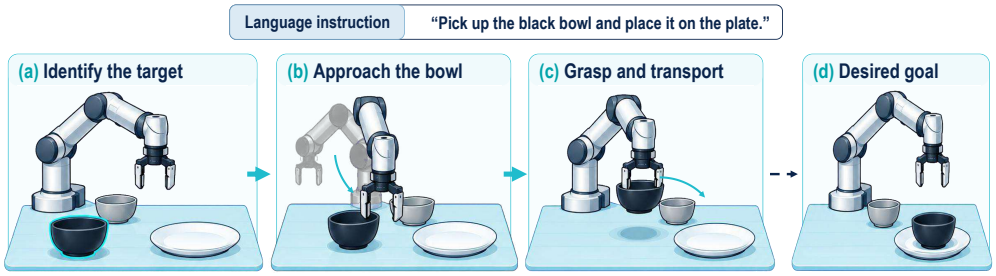


Fig. 1

Despite its strong performance in generalist robot manipulation and parameter efficient adaptation, OpenVLA explicitly acknowledges that the current model supports only single image observations , cannot directly incorporate multiple images[12]. Moreover, the standard OpenVLA training objective mainly supervises the prediction of action tokens from expert demonstrations, cannot distinguish successful trajectories from failed trajectories or learning their relative preferences to know which action is better. While its deployment pipeline lacks dedicated mechanisms for runtime anomaly monitoring, task progress assessment and failure recovery, this makes OpenVLA more like a open loop model, when perception or control errors move the robot away from the expected trajectory, the original policy cannot reliably identify the anomaly or actively return to a recoverable state. Therefore, OpenVLA still leaves room for improvement in temporal information modeling, the use of successful and failed experiences and recovery from execution anomalies. As illustrated in Fig. 1, robotic manipulation is guided by a language instruction.

To address these limitations, we propose TPRVLA, which extends the original OpenVLA through temporal representation, preference learning, and execution recovery. Inspired by temporal adaptation and robot memory research[17][18], we introduce a lightweight temporal memory adapter that incorporates consecutive visual observations without modifying the original OpenVLA backbone. The adapter aggregates consecutive observations and extracts changes between adjacent frames, then forms short term memory through a gated recurrent unit, and injects the resulting information into the current visual features through a gated residual connection. During training, the OpenVLA backbone remains frozen and only the temporal adapter is updated, reducing the training cost while preserving the capabilities of the original policy[19]. Based on preference optimization[20], we then construct preference pairs from successful and failed trajectories and combine DPO loss, which increases the relative probability of successful action sequences,with BC on successful trajectories to reduce policy drift, while using LoRA for parameter efficient policy adaptation[16]. Finally, drawing on runtime failure detection and recovery research[10][21][22], we design a task independent state monitor that identifies persistent execution anomalies from physical progress and visual changes, then we propose checkpoint guided closed loop recovery, allowing the policy to continue execution from the corrected state. Once the anomaly evidence satisfies the triggering condition for several consecutive steps, the system selects either a pregrasp anchor or a recent state with valid progress as

a recovery checkpoint. After recovery, the previous temporal buffer is cleared so that the policy can rebuild its observation history from the corrected state.

The main contributions of this work are summarized as follows:

- We propose a lightweight temporal memory adapter that injects short term motion information from consecutive observations into the frozen OpenVLA backbone.
- We introduce a preference optimization strategy that learns from automatically paired successful and failed trajectories using a joint DPO and behavior cloning objective.
- We develop a task independent state monitor with checkpoint guided closed loop recovery to detect persistent execution anomalies and resume control from a recoverable state.
- Extensive evaluations comprising 900 rollouts across ten LIBERO Spatial tasks with three random seeds, together with ablation and efficiency analyses, validate the effectiveness of all three modules while showing less than 1% additional peak GPU memory.

2 Related Work

2.1 Vision-Language-Action Models for Robotic Manipulation

Research on language conditioned robotic manipulation has progressed from high-level semantic planning to direct action generation. SayCan grounds language model plans through robotic affordance evaluation, while PaLM-E incorporates continuous visual and state information into an embodied multimodal language model[1][2]. ReAct further demonstrates the value of interleaving reasoning and acting when an agent interacts with an environment[23]. Moving from high level planning to low level control, RT-1 formulates robotic manipulation as large scale sequence modeling, and RT-2 further advances the VLA paradigm by jointly learning from web-scale vision language data and robot trajectories and representing robot actions as tokens[3][4]. The release of Open X-Embodiment further provides heterogeneous trajectories collected across tasks and robot platforms, laying the data foundation of knowledge transfer [5]. Based on this development, Octo learns a generalist policy from cross embodiment data, π_0 and RDT-1B generate continuous actions through flow or diffusion models, while RoboFlamingo adapts vision language foundation models for robot imitation[24][25][26][27]. Other studies improve task conditioning and action organization through multimodal prompts and language based action hierarchies[28][29]. Hori et al. [30] combined active clarification and passive modification to improve long-horizon offline task planning. Hentosh et al. [31] developed a depth-augmented diffusion policy that fuses RGB and depth features, using predicted pseudo-depth as an alternative to sensor depth. These studies demonstrate the transition of robot learning from specialized policies toward generalist models that jointly connect perception, language, and control. However, these research works lack relevant utilization of temporal visual information and experience of failure. Moreover, these works have not proposed a solution to the problem of gradually accumulated robot errors in long-horizon tasks.

Among existing VLAs, OpenVLA provides the most relevant open source foundation for our work. It constructs a 7B VLA model upon Prismatic VLMs, combines the spatial visual features of DINOv2 with the semantic visual features of SigLIP and converts continuous robot actions into tokens that can be generated by a language model[12][13][14][15]. Together with diverse robot datasets and benchmarks such as BridgeData V2, DROID, and LIBERO, OpenVLA provides a reproducible platform for studying task transfer and downstream policy adaptation[32][7][6]. Its support for LoRA and parameter efficient adaptation further enables model improvement under limited computational resources[16]. However, OpenVLA only accepts the single image and does not explicitly retain observation history, leaving short term motion and task progress insufficiently represented. Its standard adaptation process mainly learns action tokens from successful demonstrations and does not explicitly exploit relative preferences between successful and failed trajectories. Moreover, OpenVLA performs reactive action prediction without dedicated mechanisms for runtime anomaly assessment and active recovery. Therefore, these limitations motivate the development of a more reliable OpenVLA-based framework that incorporates temporal context, learns from both success and failed experience, and supports lightweight runtime correction.

2.2 Temporal Modeling and Memory for Robot Policies

Temporal modeling enables robots to understand state transitions and task progress from consecutive observations. R3M learns transferable temporal visual representations from egocentric videos[33]. AIM extends static visual representations to video understanding by inserting lightweight temporal adapters into a frozen image model[17]. RoboFlamingo models sequential observations with a recurrent policy head for language conditioned robotic manipulation[27]. MResT combines visual information at different spatial and temporal resolutions to balance global task understanding with local control feedback[34]. MemoryVLA preserves and retrieves historical information through a perceptual cognitive memory bank to improve state understanding in long-horizon manipulation[18]. However, existing temporal modeling methods for robot policies commonly rely on separate policy heads, multiresolution perception networks, or complex memory modules, introducing additional parameters and computational overhead. A lightweight approach that incorporates observation history into OpenVLA with few trainable parameters remains insufficiently explored.

2.3 Direct Preference Optimization for Robot Policies

Direct Preference Optimization increases the likelihood of preferred samples over rejected samples while constraining deviation from a reference model, enabling preference alignment without an explicit reward model or reinforcement learning[20]. For robot policy adaptation, successful and failed trajectories can form preference pairs that allow the model to learn their relative action differences directly. HAPO further refines robot policies by constructing action preferences from deployment failures and human corrections[35]. However, existing robot preference learning methods often rely

on human corrections or manual annotations. This dependence increases data preparation costs, while the automatic selection of reliable preference pairs from robot rollout trajectories remains insufficiently explored.

2.4 Runtime Failure Detection and Recovery for Robot Policies

For runtime monitoring and recovery, Inner Monologue uses success detection and environment feedback for closed loop task replanning[11]. Sentinel monitors failures in generative robot policies by evaluating action consistency and task progress[10]. AHA uses a vision language model to detect manipulation failures and reason about their causes[36]. REFLECT summarizes robot experiences to generate failure explanations and corrective plans[21]. FOREWARN predicts the future outcomes of candidate actions and uses a vision language model for runtime policy selection[37]. However, existing runtime approaches commonly rely on additional vision language models, predictive models, or complex reasoning and replanning procedures, resulting in considerable computational overhead and response latency during deployment. Lightweight monitoring and recovery mechanisms that can operate efficiently with VLAs remain insufficiently explored.

3 Preliminaries

3.1 Problem Formulation and Research Objective

We study the problem of language-conditioned robotic manipulation with pretrained VLA models. According to a given natural language instruction, robot must continuously interpret its visual observations and generate control commands to complete the task successfully. Following OpenVLA[12], at time step t, the policy receives an RGB observation $\mathbf{o}_t \in \mathbb{R}^{H\times W\times 3}$ and a language instruction $\mathbf{l}$, and predicts a seven-dimensional robot action

$$\mathbf{a}_t = [\Delta\mathbf{p}_t, \Delta\boldsymbol{\rho}_t, g_t] \in \mathbb{R}^7, \tag{1}$$

where $\Delta\mathbf{p}_t \in \mathbb{R}^3$ denotes the Cartesian displacement of the end effector, $\Delta\boldsymbol{\rho}_t \in \mathbb{R}^3$ represents the corresponding rotational displacement, and g_t specifies the gripper command. Executing $\mathbf{a}_t$ changes the underlying environment state according to the unknown transition dynamics

$$\mathbf{s}_{t+1} \sim \mathcal{P}\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right). \tag{2}$$

A language-conditioned manipulation task is therefore defined by an instruction $\mathbf{l}$, an initial state $\mathbf{s}_1$, and a set of goal states $\mathcal{G}(\mathbf{l})$. The task is considered successful if the robot reaches a state in $\mathcal{G}(\mathbf{l})$ within a finite execution horizon T.

The original OpenVLA policy formulates robot control as single-step action prediction, mapping the current observation and instruction directly to an action:

$$\hat{\mathbf{a}}_t = \pi_{\mathrm{VLA}}\left(\mathbf{o}_t, \mathbf{l}\right). \tag{3}$$

Although the policy can be queried again after each executed action, each prediction is produced without explicitly modeling the preceding visual context. This formulation is insufficient for long-horizon manipulation because a single RGB observation provides only a partial description of the current environment state. In particular, the current image may not reveal the motion trend of the end effector. Consequently, two visually similar observations may correspond to different interaction histories and require different subsequent actions:

$$\mathbf{o}_t^{(i)} \approx \mathbf{o}_t^{(j)}, \qquad \mathbf{a}_t^{*(i)} \neq \mathbf{a}_t^{*(j)}. \tag{4}$$

To address these limitations, our objective is to transform OpenVLA into a temporally aware and failure-recoverable robotic control policy while preserving its pretrained visual and semantic knowledge. Given the recent visual history

$$\mathcal{H}_t = \left\{\mathbf{o}_{\max(1,t-K+1)}, \ldots, \mathbf{o}_t\right\}, \tag{5}$$

we aim to learn a history-conditioned policy

$$\pi_\Theta\left(\mathbf{a}_t \mid \mathcal{H}_t, \mathbf{l}\right) \tag{6}$$

that maximizes the probability of successfully completing the instructed task:

$$\Theta^* = \arg\max_\Theta \mathbb{E}_{\tau\sim\pi_\Theta}\left[\mathcal{S}\left(\tau, \mathbf{l}\right)\right], \tag{7}$$

where $\mathcal{S}(\tau, \mathbf{l}) \in \{0, 1\}$ indicates whether trajectory τ successfully completes instruction $\mathbf{l}$. To achieve this objective, TPRVLA introduces three complementary capabilities: temporal context modeling for coherent action prediction, preference learning and closed-loop recovery for correcting execution failures. The framework is designed to improve long-horizon task success and robustness while introducing only limited additional trainable parameters and computational overhead.

3.2 Temporal modeling

Temporal information is essential for understanding dynamic visual observations. Although a single image describes the spatial configuration of a scene, it cannot directly reveal motion direction or interaction progress. This limitation is particularly important in robotic manipulation, where the appropriate action may depend on both the current observation and the preceding visual states [18].

A parameter efficient solution is to insert lightweight bottleneck adapters into a frozen image transformer [17]. Given an input feature $\mathbf{X} \in \mathbb{R}^{L\times D}$, an adapter projects the feature into a compact representation and then restores its original dimension:

$$\text{Adapter}(\mathbf{X}) = \sigma\left(\mathbf{X}\mathbf{W}_{\text{down}} + \mathbf{b}_{\text{down}}\right)\mathbf{W}_{\text{up}} + \mathbf{b}_{\text{up}}, \tag{8}$$

where $\mathbf{W}_{\text{down}} \in \mathbb{R}^{D \times d_b}$, $\mathbf{W}_{\text{up}} \in \mathbb{R}^{d_b \times D}$, and $d_b \ll D$. Only the adapter parameters need to be optimized, thereby reducing the number of trainable parameters while retaining the visual knowledge of the pretrained backbone [19].

For T consecutive frames, the extracted visual features are represented as $\mathbf{Z} \in \mathbb{R}^{T \times (N+1) \times D}$. Temporal modeling rearranges these features according to

$$\mathbf{Z}^{\text{temp}} = \mathcal{R}_t(\mathbf{Z}) \in \mathbb{R}^{(N+1) \times T \times D}, \tag{9}$$

so that tokens at the same spatial location form a temporal sequence. Temporal self attention is then applied along the frame dimension. In AIM, the attention parameters are reused from the frozen spatial attention layer, while the adapter learns the temporal adjustment [17]:

$$\widehat{\mathbf{Z}}_l = \mathcal{R}_t^{-1}\Big(\mathbf{Z}_{l-1}^{\text{temp}} + \text{Adapter}\left(\text{T-MSA}(\text{LN}(\mathbf{Z}_{l-1}^{\text{temp}}))\right)\Big). \tag{10}$$

This operation allows the visual encoder to capture object displacement, interaction continuity, and changes in scene state across frames without training an additional temporal encoder from scratch.

3.3 Direct Preference Optimization

Conventional preference alignment based on reinforcement learning generally consists of two stages [20]. First, a reward model $r_\phi(\mathbf{x}, \mathbf{y})$ is trained from a preference dataset $\mathcal{D}_{\text{pref}} = \{(\mathbf{x}, \mathbf{y}_w, \mathbf{y}_l)\}$, where $\mathbf{y}_w$ and $\mathbf{y}_l$ denote the preferred and rejected outputs for input $\mathbf{x}$. The preference probability is modeled as

$$p_\phi\left(\mathbf{y}_w \succ \mathbf{y}_l \mid \mathbf{x}\right) = \sigma\Big(r_\phi(\mathbf{x}, \mathbf{y}_w) - r_\phi(\mathbf{x}, \mathbf{y}_l)\Big). \tag{11}$$

The learned reward model is subsequently used to optimize the policy through reinforcement learning. The policy objective maximizes the predicted reward while constraining the policy from deviating excessively from a reference policy:

$$\begin{aligned} \max_{\pi_\theta} \quad & \mathbb{E}_{\mathbf{y} \sim \pi_\theta(\cdot|\mathbf{x})}\left[r_\phi(\mathbf{x}, \mathbf{y})\right] \\ & - \beta D_{\text{KL}}\left(\pi_\theta(\cdot \mid \mathbf{x}) \,\|\, \pi_{\text{ref}}(\cdot \mid \mathbf{x})\right), \end{aligned} \tag{12}$$

where β determines the strength of the constraint. This procedure requires separate reward model training and policy optimization. It also relies on outputs sampled from the current policy, which increases computational complexity and allows errors in the reward model to affect policy learning.

DPO simplifies this procedure by expressing the implicit reward directly through the likelihood ratio between the trainable policy and the reference policy:

$$r_\theta(\mathbf{x}, \mathbf{y}) = \beta \log \frac{\pi_\theta(\mathbf{y} \mid \mathbf{x})}{\pi_{\text{ref}}(\mathbf{y} \mid \mathbf{x})}. \tag{13}$$

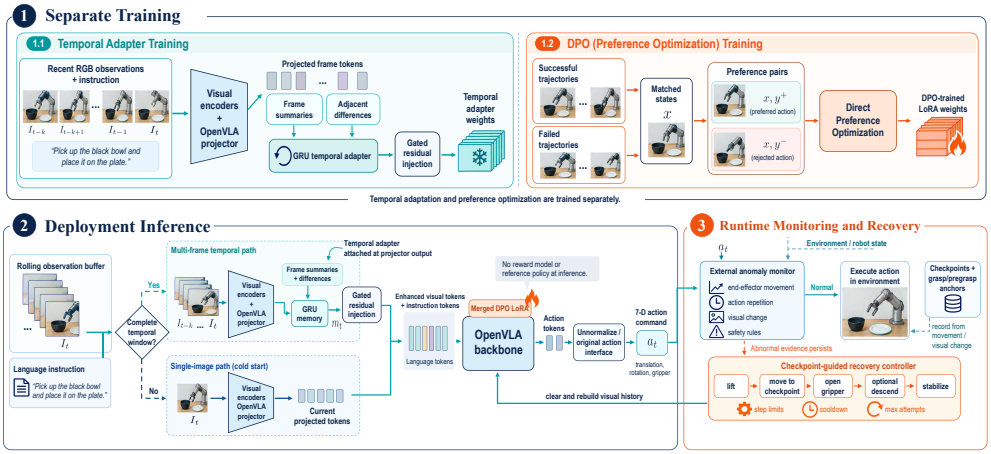


Fig. 2

The policy can then be optimized using the following preference classification objective:

$$\mathcal{L}_{\mathrm{DPO}}(\theta) = -\mathbb{E}_{\mathcal{D}_{\mathrm{pref}}}\Big[\log \sigma\big(r_\theta(\mathbf{x}, \mathbf{y}_w) - r_\theta(\mathbf{x}, \mathbf{y}_l)\big)\Big]. \tag{14}$$

Minimizing this objective increases the relative probability of preferred outputs while suppressing rejected outputs. Compared with the conventional reinforcement learning procedure, DPO removes the need for a separate reward model, repeated policy sampling, and policy gradient optimization. It therefore incorporates preference information through a simpler supervised objective while retaining the regularization provided by the reference policy [20].

4 Methodology

4.1 Overall Framework

Our framework extends OpenVLA with three complementary components: a lightweight temporal memory adapter, direct preference optimization, and an external runtime anomaly monitor with checkpoint guided recovery. The Temporal Adapter incorporates recent visual observations into action prediction through recurrent memory and gated residual injection. Direct preference optimization adapts the policy using paired actions constructed from matched states in successful and failed trajectories. The runtime monitor identifies execution anomalies through measurable visual and physical signals and activates a recovery controller when intervention is required. Temporal adaptation and preference optimization are trained separately, and their resulting parameters are combined during deployment. The overall framework of TPRVLA is illustrated in Fig. 2.

During normal execution, the system maintains a rolling buffer of RGB observations together with the language instruction. Once the buffer contains a complete temporal window, each observation is processed by the visual encoders and the original OpenVLA projector. The Temporal Adapter aggregates frame summaries and adjacent frame differences through a gated recurrent unit (GRU), then injects the resulting memory into the projected tokens of the current observation. These enhanced tokens are combined with the instruction using the original OpenVLA multimodal input layout. The DPO trained LoRA weights are merged into the backbone before inference, allowing the adapted model to generate action tokens directly without an additional reward model or reference policy. The tokens are decoded and unnormalized using the original action interface to obtain a seven dimensional command containing end effector translation, rotation, and gripper control. Before a complete history window is available, the model follows its single image prediction path.

Before each proposed action is executed, the external monitor checks for execution stagnation and unsafe motion using end effector movement, action repetition, visual change, and predefined safety conditions. These checks are implemented as numerical rules and do not require an additional vision language model. The system records robot state snapshots when physical movement or visual change exceeds a threshold and maintains grasp related anchors for recovery. When sufficient abnormal evidence persists, a bounded recovery controller temporarily replaces the policy action. It lifts the end effector, moves toward a selected checkpoint position at the elevated height, and when a grasp related anchor is used, opens the gripper and optionally descends toward the stored pregrasp height. After stabilization, control returns to OpenVLA and the visual history buffer is cleared and rebuilt from subsequent observations. Recovery phases have step limits, and repeated interventions are constrained by a cooldown period and a maximum attempt budget. Completion of the recovery sequence returns control to the policy; task success is determined by the benchmark environment.

4.2 Lightweight Temporal Memory Adapter

OpenVLA predicts actions from the current RGB observation and language instruction [12]. Although a single observation provides information about objects and spatial relationships, it does not explicitly describe recent scene changes. Visually similar observations may therefore correspond to different interaction histories and require different actions. To incorporate this missing context, we introduce a Temporal Adapter that summarizes a short observation history and uses it to modify the current visual representation.

AIM demonstrates the value of adapting pretrained visual models through additional trainable modules while freezing the original backbone [17]. Our design introduces a compact representation of observation history that separates the visual state of each frame from its change relative to the preceding frame. Frame summaries retain a compressed description of historical scene content, while adjacent frame differences explicitly represent recent visual changes. These complementary components are encoded through separate projection branches and aggregated chronologically by a GRU. The resulting memory is injected into the current projected visual tokens

through a learned gate, incorporating historical context while preserving the original visual token count.

Construction of the visual history

At time step t, the visual history is defined as the ordered sequence

$$\mathcal{H}_t = (\mathbf{I}_{t-T+1}, \ldots, \mathbf{I}_t)\,, \tag{15}$$

where T is the temporal window size and $\mathbf{I}_t$ is the current RGB observation. The buffer is updated by appending the latest observation and removing the oldest one when its capacity is exceeded. Temporal processing is activated only after T observations have been collected. This condition also applies after the history buffer is cleared following recovery.

Let $\mathcal{E}_{\mathrm{D}}$ and $\mathcal{E}_{\mathrm{S}}$ denote the DINOv2 and SigLIP visual encoders. Their patch features are concatenated and mapped into the language embedding space by the OpenVLA projector $\mathcal{P}$:

$$\begin{aligned} \mathbf{F}_\tau &= [\mathcal{E}_{\mathrm{D}}(\mathbf{I}_\tau) \,\|\, \mathcal{E}_{\mathrm{S}}(\mathbf{I}_\tau)]\,, \\ \mathbf{P}_\tau &= \mathcal{P}(\mathbf{F}_\tau) \in \mathbb{R}^{N\times D}, \end{aligned} \tag{16}$$

where $\|$ denotes feature concatenation, N is the number of patch tokens, and D is the language embedding dimension. The same encoders and projector process every observation in the window, producing

$$\mathbf{P}_t^{\mathrm{seq}} = \mathrm{Stack}\,(\mathbf{P}_{t-T+1}, \ldots, \mathbf{P}_t) \in \mathbb{R}^{T\times N\times D}. \tag{17}$$

The Temporal Adapter receives this sequence after visual projection.

Frame summaries and temporal differences

To retain both historical visual states and their evolution, we represent each observation using two complementary components: a frame summary and an adjacent frame difference. The frame summary compresses the projected visual tokens through spatial averaging:

$$\mathbf{s}_{t,j} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{P}_{t-T+j,n} \in \mathbb{R}^D, \qquad j = 1, \ldots, T, \tag{18}$$

where j indexes observations within the current temporal window. Each summary retains a compact representation of the scene observed at that time step. Processing these summaries in chronological order provides the temporal module with information about the preceding visual states.

We additionally compute the difference between consecutive summaries:

$$\boldsymbol{\delta}_{t,j} = \begin{cases} \mathbf{0}, & j = 1, \\ \mathbf{s}_{t,j} - \mathbf{s}_{t,j-1}, & j > 1. \end{cases} \tag{19}$$

The first difference is set to zero because no preceding observation is available within the window. The difference component explicitly represents changes in the summarized

visual state, providing a direct signal of how the scene evolves between adjacent observations. Retaining the frame summary alongside this difference supplies the context needed to interpret the change.

The two components are encoded through separate projection branches and then concatenated:

$$\mathbf{z}_{t,j} = [\phi_s(\mathbf{s}_{t,j}) \,\|\, \phi_\delta(\boldsymbol{\delta}_{t,j})] \in \mathbb{R}^{2d}, \tag{20}$$

where d is the memory dimension. Each branch consists of a linear projection, layer normalization, GELU activation, and optional dropout. Separate encoding allows the two branches to learn representations for historical scene content and temporal variation, respectively. Their concatenation supplies the subsequent GRU with both types of information at every time step, enabling it to summarize the ordered observation history into a compact recurrent memory.

GRU temporal aggregation

The concatenated features are processed chronologically by a GRU:

$$\mathbf{h}_{t,j} = \mathrm{GRU}\left(\mathbf{z}_{t,j}, \mathbf{h}_{t,j-1}\right), \qquad \mathbf{h}_{t,0} = \mathbf{0}, \tag{21}$$

where $\mathbf{h}_{t,j} \in \mathbb{R}^d$ is the recurrent state at position j in the window. The final state $\mathbf{h}_{t,T}$ summarizes the ordered sequence of frame representations and temporal differences.

The recurrent state is initialized to zero and recomputed for each input window. Thus, the effective memory is restricted to the observations retained in the rolling buffer. The final state is projected back to the language embedding dimension:

$$\mathbf{m}_t = \mathrm{LN}\left(\mathbf{W}_m \mathbf{h}_{t,T} + \mathbf{b}_m\right) \in \mathbb{R}^D, \tag{22}$$

where LN denotes layer normalization. The resulting memory vector is shared across the spatial tokens of the current observation.

Gated residual injection

A learned gate regulates the contribution of the memory to each current token:

$$\mathbf{g}_{t,n} = \mathrm{sigmoid}\left(\mathbf{W}_g \left[\mathbf{P}_{t,n} \,\|\, \mathbf{m}_t\right] + \mathbf{b}_g\right) \in (0,1)^D. \tag{23}$$

Although all tokens receive the same memory vector, their gates depend on their individual current features. This allows the contribution of the temporal memory to vary across tokens and feature channels.

The enhanced visual token is

$$\widetilde{\mathbf{P}}_{t,n} = \mathbf{P}_{t,n} + \lambda_{\mathrm{temp}}\left(\mathbf{g}_{t,n} \odot \mathbf{m}_t\right), \tag{24}$$

where $\odot$ denotes elementwise multiplication and $\lambda_{\mathrm{temp}} \geq 0$ is a fixed scalar controlling the overall injection strength. The scalar is a configurable hyperparameter, while the gate is learned during temporal adaptation.

Setting $\lambda_{\text{temp}} = 0$ removes the temporal contribution and recovers the current projected representation of the loaded backbone. Applying the residual update to all current tokens produces

$$\widetilde{\mathbf{P}}_t \in \mathbb{R}^{N \times D}. \tag{25}$$

Only these N enhanced tokens are supplied to the language model, preserving the visual token count of the original single image interface.

Action prediction and optimization

The enhanced tokens are inserted into the instruction prompt embeddings using the original OpenVLA multimodal sequence layout. Let $\mathbf{y}_t = (y_{t,1}, \ldots, y_{t,K})$ denote the action token sequence. The policy generates these tokens autoregressively:

$$p_\theta\left(\mathbf{y}_t \mid \mathcal{H}_t, \mathbf{l}\right) = \prod_{k=1}^{K} p_\theta\left(y_{t,k} \mid y_{t,<k}, \widetilde{\mathbf{P}}_t, \mathbf{l}\right), \tag{26}$$

where $\mathbf{l}$ is the language instruction. In the evaluated manipulation setting, $K = 7$, corresponding to the translation, rotation, and gripper action dimensions. The generated tokens are converted into continuous commands through the original OpenVLA decoding and action unnormalization procedures.

During temporal training, the visual encoders, visual projector, and language model remain frozen. The Temporal Adapter is optimized using the original supervised action prediction loss, with trainable parameters

$$\theta_{\text{temp}} = \{\theta_s, \theta_\delta, \theta_{\text{GRU}}, \theta_m, \theta_g\}, \tag{27}$$

covering the two input projection branches, the GRU, the memory output projection, and the injection gate, including their associated normalization parameters. The residual strength λ_{temp} is not learned. DPO is performed separately using LoRA adaptation of the base policy, and its weights are merged into the backbone for combined evaluation with the trained Temporal Adapter.

This design introduces a bounded visual history into action prediction while retaining the original action interface and a fixed number of language model input tokens. Temporal aggregation is performed on frame summaries through a compact recurrent state. Encoding the observation history introduces additional computation, whose deployment cost is assessed in the efficiency analysis.

4.3 Failure Informed Preference Optimization

Learning from successful demonstrations alone does not explicitly compare alternative actions associated with successful and failed executions. To incorporate information from failures, we construct local action preferences from the original policy's rollouts and use them to adapt OpenVLA. Based on the DPO objective [20], our method combines conservative state matching, confidence based balanced sampling, and behavior cloning on successful observations. These components account for differences between the source states of paired actions while providing explicit preference supervision.

Algorithm 1 Temporal Memory Extraction and Gated Fusion

Require: Projected visual tokens $Z_{1:T}$, trained adapter parameters ϕ, residual scale ρ
Ensure: Enhanced current tokens $\widetilde{Z}_T$

$\bar{z}_k \leftarrow \mathrm{MeanPool}(Z_k)$ for $k = 1, \ldots, T$
$\delta_1 \leftarrow \mathbf{0}$; $\delta_k \leftarrow \bar{z}_k - \bar{z}_{k-1}$ for $k = 2, \ldots, T$
$h_0 \leftarrow \mathbf{0}$
for $k = 1, \ldots, T$ **do**
    $e_k \leftarrow [f_{\mathrm{s}}(\bar{z}_k); f_{\mathrm{d}}(\delta_k)]$
    $h_k \leftarrow \mathrm{GRU}_\phi(e_k, h_{k-1})$
end for
$R \leftarrow \mathrm{Broadcast}(f_{\mathrm{m}}(h_T))$
$G \leftarrow g_\phi([Z_T; R])$ ▷ Sigmoid gate
$\widetilde{Z}_T \leftarrow Z_T + \rho\, G \odot R$
return $\widetilde{Z}_T$

Construction of action preferences

We collect successful and failed trajectories using the original OpenVLA policy, with temporal adaptation, preference adaptation, and runtime recovery disabled. Each trajectory records visual observations, end effector positions, model actions, the language instruction, and the final task outcome. The recorded model actions are retained before conversion to the environment's gripper convention.

For a sampled state from a failed trajectory, we search for a matching state in successful trajectories of the same task. Let $(\mathbf{I}_i^-, \mathbf{p}_i^-, \mathbf{a}_i^-)$ denote the observation, end effector position, and action of the failure sample, and let $(\mathbf{I}_j^+, \mathbf{p}_j^+, \mathbf{a}_j^+)$ denote a candidate success sample. For an observation at index u in a trajectory of length L, we define its normalized trajectory position as

$$\rho = \frac{u}{\max(1, L-1)}. \tag{28}$$

This quantity provides a temporal matching proxy derived from trajectory indices.

Candidate matching considers visual distance, end effector distance, normalized trajectory position difference, and action distance:

$$\begin{aligned} d_{\mathrm{vis}}(i,j) &= \frac{1}{d_f} \left\| f(\mathbf{I}_i^-) - f(\mathbf{I}_j^+) \right\|_1, \\ d_{\mathrm{pos}}(i,j) &= \left\| \mathbf{p}_i^- - \mathbf{p}_j^+ \right\|_2, \\ d_{\mathrm{time}}(i,j) &= \left| \rho_i^- - \rho_j^+ \right|, \\ d_{\mathrm{act}}(i,j) &= \left\| \mathbf{a}_i^- - \mathbf{a}_j^+ \right\|_2, \end{aligned} \tag{29}$$

where $f(\cdot) \in \mathbb{R}^{d_f}$ is a flattened RGB representation obtained by spatial average pooling of normalized images. The action distance is computed using the recorded model outputs.

An eligible candidate must satisfy

$$\begin{aligned} d_{\text{vis}}(i,j) &\le \eta_{\text{vis}}, \\ d_{\text{pos}}(i,j) &\le \eta_{\text{pos}}, \\ d_{\text{time}}(i,j) &\le \eta_{\text{time}}, \\ d_{\text{act}}(i,j) &\ge \eta_{\text{act}}, \end{aligned} \tag{30}$$

where the first three thresholds restrict contextual differences and the last threshold excludes nearly identical action pairs. Failure observations are examined at a fixed stride. Successful sample reuse and the number of pairs contributed by each failed trajectory are also bounded.

Among the eligible candidates $\mathcal{C}_i$, we select the one with the smallest normalized matching distance:

$$\begin{aligned} S_{ij} &= \frac{d_{\text{vis}}(i,j)}{\eta_{\text{vis}}} + \frac{d_{\text{pos}}(i,j)}{\eta_{\text{pos}}} + \frac{d_{\text{time}}(i,j)}{\eta_{\text{time}}}, \\ j^* &= \arg\min_{j \in \mathcal{C}_i} S_{ij}. \end{aligned} \tag{31}$$

Samples without an eligible match are discarded. The selected success action becomes the preferred action $\mathbf{a}_i^+ = \mathbf{a}_{j^*}^+$, while $\mathbf{a}_i^-$ becomes the rejected action. These labels are inferred from trajectory outcomes and therefore represent approximate local preferences.

Pair confidence and balanced sampling.

The confidence assigned to each accepted pair is

$$c_i = \text{clip}\left(1 - \frac{S_{ij^*}}{3}, c_{\min}, 1\right), \tag{32}$$

where $c_{\min} > 0$ is a confidence floor. A smaller matching distance produces a higher confidence.

Pairs are divided into training and validation sets according to their source failure episodes, keeping all pairs from the same failure episode in a single split. Within the training set, sampling balances tasks and failure episodes, while confidence determines the relative probability of pairs within an episode:

$$P(i) = \frac{1}{|\mathcal{Q}|} \frac{1}{|\mathcal{E}_{q_i}|} \frac{c_i}{\sum_{j \in \mathcal{D}_{q_i,e_i}} c_j}. \tag{33}$$

Here, $\mathcal{Q}$ denotes the represented training tasks, $\mathcal{E}_{q_i}$ denotes the failure episodes contributing training pairs to task q_i, and $\mathcal{D}_{q_i,e_i}$ contains the training pairs from episode e_i of that task. Sampling is performed with replacement. This distribution prevents tasks or episodes with more pairs from dominating the sampling probability, while favoring closer matches within each episode.

Preference comparison under a shared observation

The preference comparison conditions both candidate actions on the failure observation and its instruction:

$$x_i^- = \left(\mathbf{I}_i^-, \mathbf{l}_i\right). \tag{34}$$

The recorded actions are normalized using the original dataset statistics and action mask, then converted into responses through the OpenVLA action tokenizer. End effector positions are used only for matching and are not appended to the policy input.

Let $\mathbf{y}_i^+$ and $\mathbf{y}_i^-$ denote the preferred and rejected responses, including their action tokens and response termination token. Their sequence log probabilities are

$$\log \pi_\theta(\mathbf{y} \mid x) = \sum_{k=1}^{|\mathbf{y}|} \log p_\theta\left(y_k \mid y_{<k}, x\right). \tag{35}$$

Instruction and visual token positions are excluded from this computation.

The preference logit measures the relative change in response probabilities with respect to the frozen reference policy:

$$z_i = \beta \left[\log \frac{\pi_\theta(\mathbf{y}_i^+ \mid x_i^-)}{\pi_{\text{ref}}(\mathbf{y}_i^+ \mid x_i^-)} - \log \frac{\pi_\theta(\mathbf{y}_i^- \mid x_i^-)}{\pi_{\text{ref}}(\mathbf{y}_i^- \mid x_i^-)}\right], \tag{36}$$

where $\beta > 0$ controls the scale of the preference logit. To account for uncertainty in preferences inferred from trajectory outcomes, we apply label smoothing:

$$\ell_i^{\text{pref}} = -(1-\epsilon)\log \sigma(z_i) - \epsilon \log \sigma(-z_i), \tag{37}$$

where σ denotes the sigmoid function and $0 \le \epsilon < 0.5$ is the smoothing coefficient. This objective encourages the preferred response relative to the rejected response under the same failure observation.

Behavior cloning on successful observations

State matching reduces contextual differences, but the preferred action still originates from another observation. We therefore retain its original successful observation $\mathbf{I}_i^+ = \mathbf{I}_{j^*}^+$ and define

$$x_i^+ = \left(\mathbf{I}_i^+, \mathbf{l}_i\right). \tag{38}$$

An additional behavior cloning term reinforces the preferred response under this original context:

$$\ell_i^{\text{BC}} = -\frac{1}{|\mathbf{y}_i^+|} \log \pi_\theta\left(\mathbf{y}_i^+ \mid x_i^+\right). \tag{39}$$

The preference term compares alternative responses under the failure observation, while the behavior cloning term preserves the association between a successful observation and its recorded action.

The effective training objective is

$$\mathcal{L}_{\text{pref}} = \mathbb{E}_{i\sim P}\left[\ell_i^{\text{pref}} + \lambda_{\text{BC}} \ell_i^{\text{BC}}\right], \tag{40}$$

Algorithm 2 Preference Optimization from Execution Outcomes

Require: Original policy π_{θ_0}, original policy rollouts $\mathcal{R}^+, \mathcal{R}^-$, coefficients $\beta, \epsilon, \lambda_{\text{BC}}$
Ensure: Trained LoRA parameters $\psi^\star$

$\mathcal{P} \leftarrow \textsc{MatchSimilarStates}(\mathcal{R}^+, \mathcal{R}^-)$
$(\mathcal{P}_{\text{tr}}, \mathcal{P}_{\text{val}}) \leftarrow \textsc{SplitByFailureEpisode}(\mathcal{P})$
Freeze θ_0 and initialize ψ
for each optimization step **do**
  $\mathcal{B} \leftarrow \textsc{BalancedSample}(\mathcal{P}_{\text{tr}})$ ▷ Use $P(i)$
  $\ell^{\psi}_{i,\pm} \leftarrow \log \pi_\psi(y_i^{\pm} \mid x_i)$ for $i \in \mathcal{B}$
  $\ell^{0}_{i,\pm} \leftarrow \log \pi_{\theta_0}(y_i^{\pm} \mid x_i)$ ▷ LoRA disabled
  $r_i \leftarrow (\ell^{\psi}_{i,+} - \ell^{\psi}_{i,-}) - (\ell^{0}_{i,+} - \ell^{0}_{i,-})$
  $\ell_i^{\text{pref}} \leftarrow -(1-\epsilon)\log \sigma(\beta r_i) - \epsilon \log \sigma(-\beta r_i)$
  $\mathcal{L}_{\text{pref}} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell_i^{\text{pref}}$
  $\mathcal{L}_{\text{BC}} \leftarrow -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{\log \pi_\psi(y_i^+ \mid x_i^+)}{|y_i^+|}$
  $\psi \leftarrow \textsc{OptimizerStep}(\psi, \mathcal{L}_{\text{pref}} + \lambda_{\text{BC}} \mathcal{L}_{\text{BC}})$
  Periodically evaluate on $\mathcal{P}_{\text{val}}$
end for
return $\psi^\star \leftarrow \psi$

where $\lambda_{\text{BC}} \geq 0$ controls the regularization strength. Pair confidence influences optimization through the balanced sampling distribution $P(i)$.

Parameter efficient adaptation and deployment

Preference optimization freezes the pretrained policy parameters and introduces LoRA updates into its eligible linear transformations. For an adapted weight matrix,

$$\mathbf{W}_\theta = \mathbf{W}_0 + \frac{\alpha}{r}\mathbf{B}\mathbf{A}, \tag{41}$$

where $\mathbf{W}_0$ is frozen, $\mathbf{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\mathbf{B} \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable matrices, r is the adaptation rank, and α controls the update scale.

Reference log probabilities are computed by temporarily disabling LoRA and evaluating the same backbone without gradient computation. This provides a fixed reference policy without storing an additional backbone copy. Preference optimization uses single observations and is trained independently of the Temporal Adapter and runtime recovery mechanism.

At deployment, the learned LoRA updates are merged into the backbone, and the resulting policy is combined with the trained Temporal Adapter and external recovery controller. The preference component therefore requires no separate reward evaluation, reference policy inference, or action distribution postprocessing during execution.

Here, x_i denotes the failure observation and instruction, whereas x_i^+ contains the matched successful observation and instruction. The sequences y_i^+ and y_i^- contain

the supervised tokens associated with the successful and failed actions, respectively. The pair confidence c_i is incorporated into the sampling distribution $P(i)$, which balances tasks and failure episodes. Training uses single-pair microbatches with gradient accumulation. The normalized confidence weighting in the implemented preference loss therefore cancels within each microbatch, making confidence affect the effective objective only through the sampling distribution $P(i)$.

4.4 Runtime Anomaly Detection and Checkpoint Guided Recovery

Temporal modeling and preference optimization improve the use of observation history and action selection, respectively, but cannot directly correct persistent stagnation or abnormal actions during execution. We therefore introduce a lightweight runtime monitoring and recovery module between policy prediction and environment execution. The module combines end effector motion, action repetition, visual change, and motion risk to identify persistent anomalies, and temporarily overrides policy actions with checkpoint guided position feedback control when necessary. It requires no additional training and evaluates observable execution anomalies.

Execution anomaly representation

Let $p_t \in \mathbb{R}^3$ denote the end effector position, $u_t \in \mathbb{R}^3$ the translation component of the policy action, and I_t the current image. Within a sliding window of length W, we compute the net end effector displacement d_t and the mean action magnitude m_t. We also construct a compact visual representation $\phi(I_t)$ through grayscale conversion, central cropping, and average pooling. The mean absolute difference between consecutive representations is temporally averaged to obtain the visual change measure $\bar{v}_t$:

$$\begin{aligned} d_t &= \|p_t - p_{t-W+1}\|_2 , \\ m_t &= \frac{1}{W} \sum_{\tau=t-W+1}^{t} \|u_\tau\|_2, \\ v_t &= \frac{1}{D} \|\phi(I_t) - \phi(I_{t-1})\|_1 , \\ \bar{v}_t &= \frac{1}{V} \sum_{\tau=t-V+1}^{t} v_\tau. \end{aligned} \tag{42}$$

Here, D is the visual representation dimension, and V is the averaging window for visual change. To identify directionally repetitive actions that produce little displacement, we further compute the mean cosine similarity c_t between adjacent translation vectors with nonnegligible norms and count the actions whose magnitudes exceed ϵ_a,

denoted by n_t. Four Boolean anomaly indicators are then defined:

$$\begin{aligned} q_t^{\text{motion}} &= \mathbb{I}[d_t < \epsilon_p \wedge m_t > \epsilon_a]\,, \\ q_t^{\text{repeat}} &= \mathbb{I}\begin{bmatrix} c_t > \eta_c \wedge d_t < 2\epsilon_p \\ \wedge\, m_t > \epsilon_a \wedge n_t \geq W - 1 \end{bmatrix}, \\ q_t^{\text{visual}} &= \mathbb{I}[\bar{v}_t < \epsilon_v \wedge \|u_t\|_2 > \epsilon_a]\,, \\ q_t^{\text{risk}} &= \mathbb{I}[p_{t,z} < z_{\min} \wedge u_{t,z} < \eta_z]\,. \end{aligned} \tag{43}$$

Here, $\mathbb{I}[\cdot]$ is the indicator function. The thresholds ϵ_p, ϵ_a, ϵ_v, and η_c control the assessment of displacement, action magnitude, visual change, and directional repetition, respectively. The height threshold $z_{\min}$ and negative action threshold η_z identify downward motion at a low end effector height. The first three indicators are enabled only after monitoring begins and their required histories are available. These signals characterize physical and visual change without equating either with semantic task progress.

Evidence fusion and persistence confirmation.

To reduce interventions caused by isolated transient signals, we combine the indicators into an anomaly score and require consecutive abnormal observations before triggering recovery:

$$\begin{aligned} s_t &= q_t^{\text{motion}} + q_t^{\text{repeat}} + q_t^{\text{visual}} + \eta_s q_t^{\text{risk}}, \\ b_t &= \mathbb{I}[s_t \geq \eta_s], \\ h_t &= \begin{cases} \min(h_{t-1} + 1, H), & b_t = 1, \\ 0, & b_t = 0. \end{cases} \end{aligned} \tag{44}$$

Here, η_s is the anomaly score threshold, and H is the persistence requirement. Motion risk alone is sufficient to reach the instantaneous score threshold, but remains subject to persistence confirmation. A new recovery procedure begins only when $h_t = H$, the cooldown has expired, the recovery attempt budget remains available, and the end effector position is accessible. This design limits both transient false triggers and the execution overhead of repeated interventions.

Checkpoint construction and selection

During normal execution, the system records a state snapshot containing the timestamp, end effector pose, and gripper information when either the displacement between consecutive observations or the visual change exceeds its corresponding threshold:

$$\chi_t = \mathbb{I}[\|p_t - p_{t-1}\|_2 \geq \epsilon_p \vee v_t \geq \epsilon_v]\,. \tag{45}$$

The system additionally establishes an initial grasp related anchor from the first qualifying gripper closing command after sufficient displacement from the initial position, and selects an earlier snapshot as a pregrasp anchor. These anchors indicate execution locations associated with grasp commands rather than verified grasp success. Recovery prioritizes the pregrasp anchor, followed by the grasp related anchor.

When neither is available, the system selects the latest sufficiently recent snapshot with adequate horizontal separation from the anomaly position, or falls back to the recent snapshot with the greatest horizontal separation. When history is insufficient, a valid last snapshot, the beginning of the motion window, or the current anomaly position provides a fallback target. Checkpoints therefore serve as positional references rather than restoring the complete environment state.

Checkpoint guided recovery

Let p_s denote the end effector position at recovery onset and p_c the selected checkpoint position. The controller first constructs a lift target and then performs a bounded horizontal return toward the checkpoint at the lifted height:

$$\begin{aligned} z_{\text{lift}} &= \max(p_{s,z}, \min(p_{s,z} + h_{\text{lift}}, z_{\max}))\,, \\ \Delta_{xy} &= \text{clipnorm}(p_{c,xy} - p_{s,xy}, d_{\max})\,, \\ p^{\text{lift}} &= \begin{bmatrix} p_{s,xy} \\ z_{\text{lift}} \end{bmatrix}, \\ p^{\text{return}} &= \begin{bmatrix} p_{s,xy} + \Delta_{xy} \\ z_{\text{lift}} \end{bmatrix}. \end{aligned} \tag{46}$$

Here, $\text{clipnorm}(\cdot, d_{\max})$ preserves the vector direction while limiting its norm to $d_{\max}$, with the distance bound determined by the checkpoint type. At each position control stage, the action is updated using the current end effector position:

$$a_t^{\text{rec}} = \begin{bmatrix} \text{clip}\big(k_p(p_t^{\text{target}} - p_t), -a_{\max}, a_{\max}\big) \\ \mathbf{0}_3 \\ g_t^{\text{rec}} \end{bmatrix}. \tag{47}$$

Here, k_p is the position feedback gain, and clip bounds each translation component independently, while rotational commands are set to zero. Recovery begins with lifting and horizontal return. A gripper opening stage is added when a grasp related anchor is selected, followed by descent to the checkpoint height when a pregrasp anchor is used. A brief stabilization stage then precedes policy handoff. The gripper command at recovery onset is retained during lifting and return, and the gripper remains open after the opening stage. Each position control stage advances upon reaching its positional tolerance or exhausting its step budget. Completion of the recovery procedure therefore does not imply that every target was reached or that the task succeeded.

Policy handoff and lightweight implementation

During recovery, controller actions replace policy actions. After recovery, the system clears the anomaly detection history and the temporal image buffer of the policy, rebuilding both from subsequent observations to prevent the recovery observation sequence from directly influencing normal policy decisions. A cooldown period and a bounded attempt budget further restrict repeated interventions. The module introduces only bounded history buffers, simple statistical operations, and position

Algorithm 3 Runtime Monitoring and Checkpoint Recovery

Require: Observation o_t, image I_t, policy action a_t^π, time t, monitor state $\mathcal{M}$
Ensure: Action a_t, updated $\mathcal{M}$, flags $b_{\text{start}}, b_{\text{handoff}}$

$a_t \leftarrow a_t^\pi$
$b_{\text{start}} \leftarrow$ false; $b_{\text{handoff}} \leftarrow$ false
if $\neg\mathcal{M}$.active **then**
    $\mathcal{M} \leftarrow \textsc{UpdateAndAssess}(\mathcal{M}, o_t, I_t, a_t^\pi, t)$
    if $\mathcal{M}$.trigger **then**
        $c \leftarrow \textsc{SelectCheckpoint}(\mathcal{M}, o_t)$
        $\mathcal{M} \leftarrow \textsc{StartRecovery}(\mathcal{M}, c, o_t, a_t^\pi)$
        $b_{\text{start}} \leftarrow$ true
    end if
end if
if $\mathcal{M}$.active **then**
    $(a_t, \mathcal{M}, b_{\text{handoff}}) \leftarrow \textsc{RecoveryStep}(\mathcal{M}, o_t)$
end if
if b_{handoff} **then**
    $\mathcal{M} \leftarrow \textsc{ClearHistoryAndStartCooldown}(\mathcal{M}, t)$
end if
return $(a_t, \mathcal{M}, b_{\text{start}}, b_{\text{handoff}})$

feedback control, without requiring an additional vision language model, reward model, or learned recovery policy. Its lightweight design concerns the additional monitoring and control machinery rather than eliminating policy inference. Recovery may increase execution length, while task success remains independently determined by the environment.

4.5 Algorithm

The temporal adapter and preference adapter are trained separately from the original OpenVLA model. Algorithm 1 describes temporal memory extraction and gated fusion. Algorithm 2 presents preference optimization from successful and failed trajectories. Algorithm 3 specifies runtime anomaly monitoring and checkpoint guided recovery. Finally, Algorithm 4 summarizes the independent training and runtime integration of these three components. The trained LoRA weights are merged into the backbone and deployed with the temporal adapter and recovery controller. During execution, persistent anomalies trigger bounded recovery when the intervention conditions are satisfied. Recovery onset may extend the execution budget, and policy handoff clears the temporal buffer. Here, b_s and b_h denote recovery onset and policy handoff, respectively.

Code availability

The code developed for this study is available from the corresponding author upon reasonable request.

Algorithm 4 Integrated Training and Execution

Require: $\pi_0, \mathcal{D}, \mathcal{R}^{\pm}, \mathcal{E}, \ell, T, L_0, B_{\text{post}}$
Ensure: Executed trajectory τ and outcome y

$\phi^\star \leftarrow \textsc{TrainTemp}(\pi_0, \mathcal{D})$
$\psi^\star \leftarrow \textsc{TrainPref}(\pi_0, \mathcal{R}^{\pm})$
$\pi_D \leftarrow \textsc{MergeLoRA}(\pi_0, \psi^\star)$
$(o_t, t) \leftarrow \textsc{ResetAndSettle}(\mathcal{E})$
$\mathcal{M} \leftarrow \textsc{InitMonitor}$
$\mathcal{B} \leftarrow \emptyset$; $\tau \leftarrow \emptyset$; $L \leftarrow L_0$; done $\leftarrow$ false
while $t < L$ **and** $\neg$done **do**
    $I_t \leftarrow \textsc{Image}(o_t)$
    $\mathcal{B} \leftarrow \textsc{AppendBounded}(\mathcal{B}, I_t, T)$
    if $|\mathcal{B}| = T$ **then**
        $\mathbf{Z}_{1:T} \leftarrow \textsc{Encode}(\pi_D, \mathcal{B})$
        $\widetilde{\mathbf{Z}}_t \leftarrow \textsc{FuseMemory}(\mathbf{Z}_{1:T}, \phi^\star)$
    else
        $\widetilde{\mathbf{Z}}_t \leftarrow \textsc{Encode}(\pi_D, I_t)$
    end if
    $a_t^\pi \leftarrow \textsc{DecodeAndConvert}(\pi_D, \widetilde{\mathbf{Z}}_t, \ell)$
    $(a_t, \mathcal{M}, b_s, b_h) \leftarrow$
     $\textsc{Monitor}(o_t, I_t, a_t^\pi, t, \mathcal{M})$
    if b_s **then**
        $r \leftarrow \textsc{StepsLeft}(\mathcal{M})$
        $L \leftarrow \max\{L, t + r + B_{\text{post}} + 1\}$
    end if
    if b_h **then**
        $\mathcal{B} \leftarrow \emptyset$
    end if
    $(o_{t+1}, \text{done}) \leftarrow \textsc{Step}(\mathcal{E}, a_t)$
    $\tau \leftarrow \tau \| (o_t, a_t, o_{t+1})$; $t \leftarrow t + 1$
end while
$y \leftarrow \textsc{Outcome}(\mathcal{E})$
return (τ, y)

5 Experiments

5.1 Datasets

We conduct all training and evaluation experiments on LIBERO, LIBERO is a language conditioned robotic manipulation benchmark containing 130 tasks organized into four task suites[6]. In our work, we use LIBERO-Spatial specifically, which contains ten tasks that share the black bowl as the same object manipulation objective but differ in the spatial relationship between the black bowl and surrounding objects. This controlled variation enables us to evaluate spatial grounding and policy robustness under different object layouts.

Following the OpenVLA data processing protocol, the Temporal Adapter is trained on the RLDS-formatted libero_spatial_no_noops dataset, we also remove near-zero actions from original demonstrations[12]. The processed dataset provides only a training split containing 432 successful human teleoperation trajectories across the ten tasks. We use all 432 trajectories to optimize the Temporal Memory Adapter for a fixed number of training steps and do not reserve an offline validation subset. Instead, we evaluate the trained temporal policy through newly executed closed loop rollouts in the LIBERO simulator, and these rollout trajectories are not added to the Temporal Memory Adapter training set. Although the OpenVLA backbone inherits knowledge learned from Open X-Embodiment, no Open X-Embodiment data are used for additional training in our experiments[5].

For preference optimization, we collect successful and failed rollouts produced by the original OpenVLA on the same task suite, with the temporal and recovery modules disabled. Based on the episode outcomes, our automatic selector constructs 268 high-confidence action preference pairs. We allocate 236 pairs to the training set and reserve the remaining 32 pairs for validation, corresponding to approximately 88.1% and 11.9% of the preference data, respectively. This process provides failure-informed supervision without requiring additional preference annotations for each frame.

The ten evaluation tasks are listed in Table 1.

5.2 Baselines

OpenVLA. We employ OpenVLA as the primary baseline because it provides the open source foundation upon which our model is developed[12]. Specifically, we use the official 7B checkpoint fine-tuned on LIBERO-Spatial without further adaptation. At each control step, the baseline receives a single RGB observation and a language instruction, then predicts a discretized seven-dimensional robot action autoregressively. OpenVLA is evaluated under the same experimental environment and evaluation protocol as TPRVLA, while the temporal memory adapter, direct preference optimization and runtime anomaly detection and checkpoint-guided recovery mechanism are disabled to preserve its original policy.

For a fair comparison, OpenVLA and our model are evaluated using the same LIBERO-Spatial tasks, language instructions, image preprocessing procedure and initial-state protocol. Therefore, their performance difference directly reflects the effects of TPRVLA.

5.3 Evaluation Metrics

Task Performance

We use task success rate as the primary metric for evaluating manipulation performance. Let K denote the number of tasks, S the number of random seeds, and $N_{i,s}$ the number of rollouts for task i under seed s. The outcome of rollout n is denoted by $y_{i,s,n} \in \{0, 1\}$, where 1 indicates that the task is successfully completed. The success rate of task i under seed s is defined as

$$SR_{i,s} = \frac{1}{N_{i,s}} \sum_{n=1}^{N_{i,s}} y_{i,s,n} \times 100\%. \tag{48}$$

The average success rate across all K tasks is calculated as

$$ASR_s = \frac{1}{K} \sum_{i=1}^{K} SR_{i,s}. \tag{49}$$

For experiments conducted with multiple random seeds, the mean and sample standard deviation of the average success rate are reported as

$$\overline{ASR} = \frac{1}{S} \sum_{s=1}^{S} ASR_s, \tag{50}$$

$$\sigma_{ASR} = \sqrt{\frac{1}{S-1}\sum_{s=1}^{S}\left(ASR_s - \overline{ASR}\right)^2}. \tag{51}$$

Computational Efficiency

We evaluate model efficiency using peak GPU memory, average per-step inference latency, and average rollout time. Given Q policy inference calls, the average per-step inference latency is defined as

$$\overline{L}_{\text{inf}} = \frac{1}{Q}\sum_{q=1}^{Q}\ell_q, \tag{52}$$

where ℓ_q denotes the latency of the q-th policy inference call. Given R evaluated rollouts, the average rollout time is calculated as

$$\overline{T}_{\text{roll}} = \frac{1}{R}\sum_{r=1}^{R}T_r, \tag{53}$$

where T_r is the control execution time of rollout r. Peak GPU memory is defined as the maximum allocated GPU memory observed during inference:

$$M_{\text{peak}} = \max_t M_{\text{allocated}}(t). \tag{54}$$

Lower values of these three metrics respectively indicate lower memory consumption, faster action generation and higher execution efficiency.

Preference Optimization Metrics

During DPO training, we evaluate different checkpoints on the validation preference set using preference accuracy and reward margin. Let π_θ denote the optimized policy, π_{ref} the frozen reference policy, x_m the input condition, and (y_m^+, y_m^-) the preferred and rejected action sequences of the m-th validation pair. The implicit reward assigned to an action sequence is defined as

$$r_\theta(x, y) = \beta\left[\log \pi_\theta(y|x) - \log \pi_{\text{ref}}(y|x)\right], \tag{55}$$

where β controls the strength of preference optimization. For M validation preference pairs, preference accuracy is calculated as

$$PA = \frac{1}{M}\sum_{m=1}^{M}\mathbb{I}\left[r_\theta(x_m, y_m^+) > r_\theta(x_m, y_m^-)\right] \times 100\%, \tag{56}$$

where $\mathbb{I}[\cdot]$ is the indicator function. The average reward margin is defined as

$$RM = \frac{1}{M}\sum_{m=1}^{M}\left[r_\theta(x_m, y_m^+) - r_\theta(x_m, y_m^-)\right]. \tag{57}$$

A higher preference accuracy indicates that the optimized policy ranks preferred actions above rejected actions more consistently, while a larger positive reward margin represents stronger separation between successful and failed actions.

5.4 Experimental Settings

Experimental Environment All experiments were conducted on a server running Ubuntu 22.04 and equipped with a single NVIDIA GeForce RTX 4090 GPU with 24 GB of memory. The software environment consisted of Python 3.10.20, CUDA 12.4, PyTorch 2.5.1, NumPy 1.26.4, and FlashAttention 2.7.3. Both model training and inference were performed using BF16 precision, with FlashAttention 2 employed to accelerate attention computation. Robotic manipulation experiments were conducted in the LIBERO simulation environment using EGL for headless rendering. The baseline model and our proposed model were trained and evaluated under the same hardware and software environment. All experimental settings were kept identical except for the rollout step limit, when recovery is triggered, TPRVLA is allowed to extend the total rollout horizon to accommodate both recovery and subsequent policy execution, which may increase the number of model inference calls and the total rollout time.

Module Training Configuration The OpenVLA backbone was frozen during Temporal Adapter training and was subsequently adapted through LoRA based DPO. The core parameters of the two trainable modules are summarized in Table 2.

Runtime Monitoring and Recovery Configuration

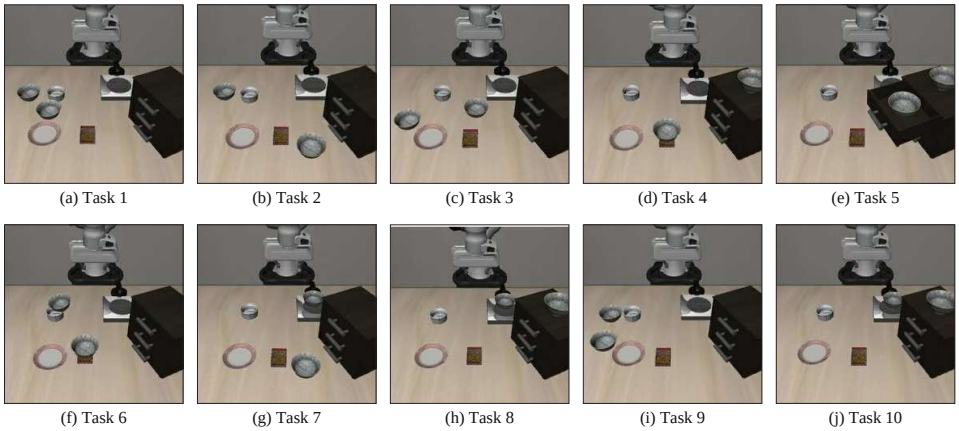
The runtime monitoring and recovery mechanism requires no additional training. Anomaly detection begins after 40 control steps, and recovery is triggered when an anomaly persists for three consecutive steps. Each episode allows at most two recovery attempts, with a cooldown interval of 15 steps between successive attempts.

5.5 Comparative Experimental Analysis

Evaluation Protocol The comparative experiments were conducted on the ten robotic manipulation tasks of the LIBERO Spatial benchmark. The model was initialized from the OpenVLA 7B checkpoint fine tuned on LIBERO Spatial, and the backbone parameters were frozen during Temporal Adapter training. The Temporal Adapter was trained using all 432 demonstrations in the dataset, corresponding to 100% of the available demonstration data, without an additional validation split. The DPO Adapter used 268 high confidence preference pairs generated by the automatic selector, of which 236 pairs were used for training and 32 pairs were used for validation, accounting for 88.1% and 11.9% of the preference data, respectively.

The OpenVLA baseline was evaluated over 30 rollouts for each task. TPRVLA was evaluated using random seeds 7, 17, and 36, with 30 rollouts per task and seed using the same initial states indexed from 0 to 29. Therefore, each task contained 90 TPRVLA rollouts when all three random seed experiments were completed. The primary evaluation metrics were the task success rate and the mean success rate across the three random seeds.

According to Table 3 TPRVLA achieved a mean success rate of 100.0% on Task7, compared with 93.3% for OpenVLA, yielding an absolute improvement of 6.7 percentage points and a relative improvement of 7.2% over the baseline. On Task10, TPRVLA achieved a mean success rate of 90.0%, whereas OpenVLA achieved 56.7%, corresponding to an absolute improvement of 33.3 percentage points and a relative improvement of 58.7%. Among the other improved tasks, TPRVLA achieved 93.3%



(a) Task 1 (b) Task 2 (c) Task 3 (d) Task 4 (e) Task 5

(f) Task 6 (g) Task 7 (h) Task 8 (i) Task 9 (j) Task 10

Fig. 3

on Task1, compared with 90.0% for OpenVLA, representing an absolute improvement of 3.3 percentage points and a relative improvement of 3.7%. On Task8, TPRVLA achieved 84.4%, compared with 76.7% for OpenVLA, corresponding to an absolute improvement of 7.7 percentage points and a relative improvement of 10.0%. On Task9, TPRVLA achieved 84.4%, exceeding the 80.0% of OpenVLA by 4.4 percentage points, equivalent to a relative improvement of 5.5%. For the tasks with degraded performance, TPRVLA achieved 93.3% on Task2, 6.7 percentage points below the 100.0% achieved by OpenVLA; 80.0% on Task3, 10.0 percentage points below the OpenVLA result of 90.0%; 92.2% on Task4, 4.5 percentage points below the OpenVLA result of 96.7%; 66.7% on Task5, 6.6 percentage points below the OpenVLA result of 73.3%; and 40.0% on Task6, 10.0 percentage points below the OpenVLA result of 50.0%. Across all ten tasks, TPRVLA achieved an overall mean success rate of 82.4%, representing an absolute improvement of 1.7 percentage points and a relative improvement of 2.1% over the OpenVLA baseline of 80.7%. For random seeds 7, 17, and 36, the average success rates across the ten tasks were 83.0%, 82.3% and 82.0%, The overall result was $(82.4 \pm 0.5)\%$.

5.6 Visualization

The initial simulation states of the ten evaluation tasks are shown in Fig. 3.

As illustrated in Fig. 4, OpenVLA abandons the task after an unsuccessful grasp attempt. In contrast, TPRVLA detects the failed interaction, initiates a new attempt, and successfully completes the grasp, which demonstrates the effectiveness of the proposed closed-loop recovery mechanism.

5.7 Ablation experiment

To evaluate the independent contribution of each proposed module, we conducted single module ablation experiments on the LIBERO-Spatial dataset and selected Task3,

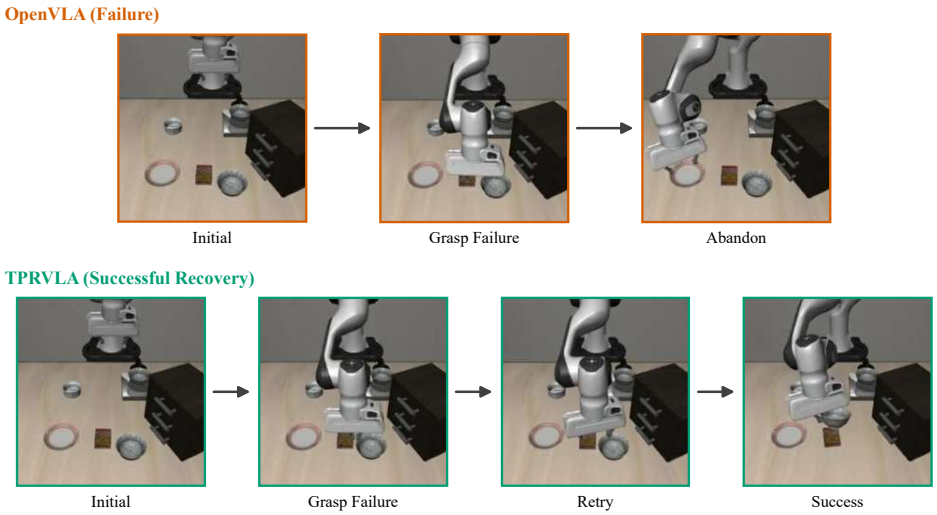


Fig. 4

Task7, Task8, and Task10 as representative tasks. These tasks cover different object initial positions, spatial manipulation relations and represent different levels of performance change achieved by TPRVLA over OpenVLA, allowing the policy to be evaluated under diverse spatial configurations. All experiments used random seed 7 and performed 30 rollouts for the 30 initial states of four tasks. Except for the module removed in each ablation, all model weights, components and evaluation settings were kept identical to those of the complete TPRVLA. Recovery-related extensions of the rollout step limit were permitted only when the runtime recovery mechanism was enabled and triggered. The complete model achieved an average success rate of 89.2% across the four tasks.

When the lightweight temporal memory adapter was removed, the success rates on Task3, Task7, Task8, and Task10 were 80.0%, 96.7%, 80.0%, and 70.0%. The average success rate decreased to 81.7%, corresponding to an absolute decrease of 7.5 percentage points and a relative decrease of 8.4% compared with the complete model. These results indicate that short term observation history and state transition information effectively complement single frame visual input, particularly for manipulation tasks that require awareness of task progress and continuous state changes.

When the direct preference optimization was removed, the success rates on the four tasks were 83.3%, 93.3%, 90.0%, and 83.3%, yielding an average success rate of 87.5%. This represents an absolute decrease of 1.7 percentage points and a relative decrease of 1.9% compared with the complete model. Although the success rates on Task3 and Task8 increased by 3.3 percentage points, those on Task7 and Task10 both decreased by 6.7 percentage points. This result suggests that preference learning constructed from successful and failed trajectories has task dependent effects, while still improving

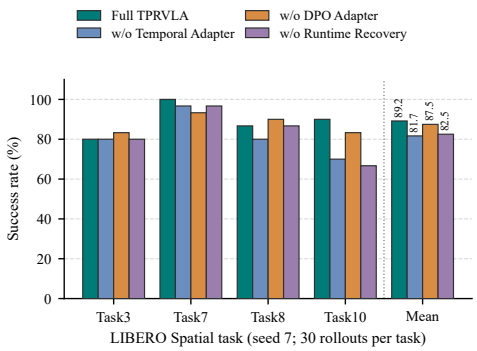


Fig. 5

the policy's overall ability to distinguish preferred actions and increasing its average performance across tasks.

When the runtime anomaly detection and checkpoint-guided recovery mechanism was removed, the success rates on Task3, Task7, Task8, and Task10 were 80.0%, 96.7%, 86.7%, and 66.7%. The average success rate decreased to 82.5%, representing an absolute decrease of 6.7 percentage points and a relative decrease of 7.5% compared with the complete model. These results indicate that the runtime anomaly detection and checkpoint-guided recovery mechanism introduces little interference during normal execution while providing effective intervention when anomalies or policy deviations occur.

As shown in Fig. 5, we evaluate the contribution of each proposed module on four representative LIBERO Spatial tasks.

5.8 Parameter Sensitivity Analysis

5.8.1 Injection strength

To analyze the sensitivity of the model to the Temporal Adapter injection strength, we adopted a one factor at a time protocol in which $t = 5$, the random seed 7, and all other experimental conditions were fixed while only the injection strength of the temporal features was varied. The task success rate on Task7 was calculated using the 1-10 initial states. When the injection strength was set to 0, 0.05 or 0.10, the model consistently achieved a success rate of 100.0%, indicating that a low injection strength does not compromise the original decision capability of backbone. When the injection strength was further increased to 0.15, 0.20, and 0.30, the success rate respectively decreased to 80.0%, 60.0% and 0.0%. These results demonstrate that the model remains stable within a low temporal feature injection range, excessive injection causes the historical observation features to interfere with the current visual representation.

As shown in Fig. 6, we analyze the sensitivity of Task7 performance to the Temporal Adapter injection strength.

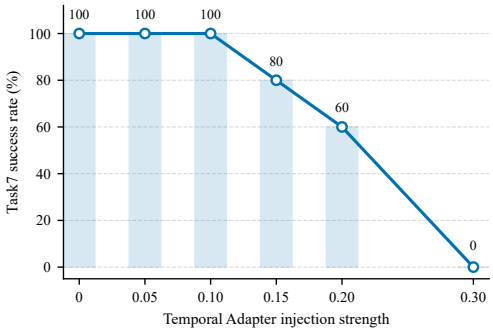


Fig. 6

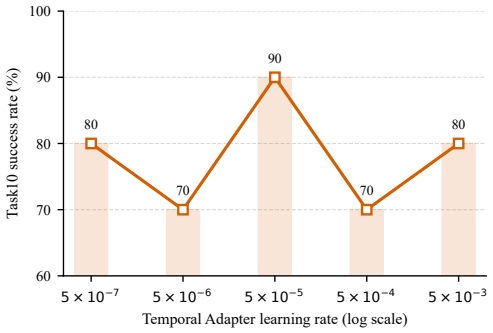


Fig. 7

5.8.2 Temporal Adapter learning rate

To analyze the sensitivity of the model to the Temporal Adapter learning rate, we varied only the learning rate while keeping the training data, parameter initialization, number of training steps, and all other experimental settings unchanged. The evaluation was conducted on Task10 using the 1-10 initial states under random seed 7. Learning rates of 5×10^{-7}, 5×10^{-6}, 5×10^{-5}, 5×10^{-4}, and 5×10^{-3} resulted in success rates of 80.0%, 70.0%, 90.0%, 70.0%, and 80.0%, respectively. These results indicate that an excessively low learning rate limits the effective optimization of the Temporal Adapter, whereas an excessively high learning rate weakens its effect by introducing overly aggressive parameter updates. Nevertheless, the model retains success rates of 70.0% to 80.0% under low or high learning rate settings, suggesting that the remaining modules continue to contribute to task execution when the contribution of the Temporal Adapter is reduced.

As shown in Fig. 7, we evaluate the sensitivity of Task10 performance to the Temporal Adapter learning rate.

5.8.3 direct preference optimization training steps

To evaluate the sensitivity of the model to the degree of preference optimization, we varied only the number of direct preference optimization training steps while keeping

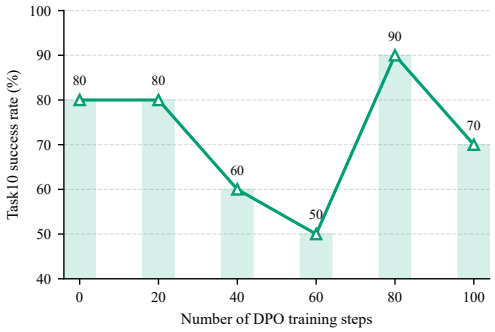


Fig. 8

the preference pairs, parameter initialization, learning rate, and all other experimental settings unchanged. The evaluation was conducted on Task10 using the 1-10 initial states under random seed 7. When the number of direct preference optimization training steps was set to 0, 20, 40, 60, 80, and 100, the corresponding success rates were 80.0%, 80.0%, 60.0%, 50.0%, 90.0% and 70.0%, respectively. These results demonstrate that insufficient optimization cannot fully exploit the relative information between successful and failed trajectories, whereas excessive optimization may cause the adapted policy to deviate from the reference policy and reduce task performance.

As shown in Fig. 8, we evaluate the sensitivity of Task10 performance to the number of direct preference optimization training steps.

5.9 Analysis of Runtime Anomaly Recovery

To examine the practical behavior of runtime anomaly detection and checkpoint-guided recovery mechanism, we analyze the formal LIBERO Spatial evaluation logs on Task1, Task3, Task7 and Task8. Each task is evaluated over 30 episodes using random seed 7 and initial state indices 0 to 29, yielding 120 episodes. An intervention episode contains at least one actual recovery intervention, whereas each recovery initiation is counted as a separate attempt. An episode is counted as a successful recovery episode when intervention occurs and the environment subsequently reports task success. Multiple attempts within the same episode do not produce multiple successful recovery counts.

Across the four selected tasks, the recovery mechanism intervenes in 20 episodes and initiates 24 recovery attempts. Of these intervention episodes, 14 ultimately succeed, corresponding to a success rate after recovery of 70.0%. Task8 provides the largest intervention sample: recovery occurs in 12 of its 30 episodes, with 15 attempts and 9 successful episodes, yielding a success rate of 75.0%. Task1 and Task3 each achieve two successful recovery episodes from three and four intervention episodes, respectively, corresponding to rates of 66.7% and 50.0%. Task7 succeeds in its single intervention episode.

As shown in Table 4, the recovery mechanism intervenes in 20 of the 120 evaluation episodes across the four selected tasks and initiates 24 recovery attempts. Fourteen

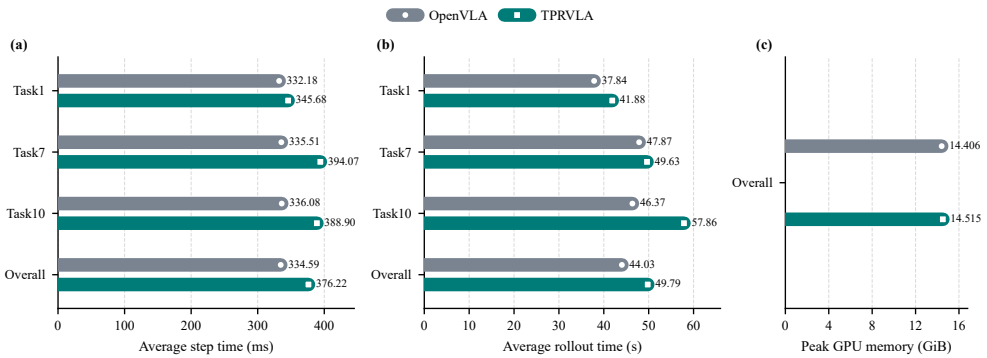


Fig. 9

intervention episodes ultimately succeed, yielding a success rate after recovery of 70.0%.

5.10 Model Efficiency Analysis

All efficiency experiments were conducted on a single NVIDIA GeForce RTX 4090 GPU. For each paired comparison, only the evaluated policy was changed between OpenVLA and TPRVLA, while the hardware and software environments, task configurations, initial state sequences, rollout settings, and evaluation protocol were kept unchanged.

On Task1, the average single step inference times of OpenVLA and TPRVLA were 332.18 ms and 345.68 ms, respectively, while their average rollout times were 37.84 s and 41.88 s. On Task7, OpenVLA required 335.51 ms per inference step and 47.87 s per rollout, whereas TPRVLA required 394.07 ms and 49.63 s, respectively. On Task10, the average single step inference time increased from 336.08 ms for OpenVLA to 388.90 ms for TPRVLA, and the average rollout time increased from 46.37 s to 57.86 s.

Across the evaluated tasks, the peak GPU memory consumption of OpenVLA and TPRVLA was 14.406 GiB and 14.515 GiB, respectively, corresponding to an increase of only 0.109 GiB. Their average single step inference times were 334.59 ms and 376.22 ms, corresponding to an increase of 12.4%, while their average rollout times were 44.03 s and 49.79 s, corresponding to an increase of 13.1%. Overall, TPRVLA achieves improved model performance with acceptable increases in GPU memory consumption and inference time.

As shown in Fig. 9, we compare the inference efficiency and GPU memory consumption of OpenVLA and TPRVLA.

6 Conclusion

This paper presented TPRVLA, an extension from OpenVLA integrating lightweight temporal memory adapter, direct preference optimization, runtime anomaly detection and checkpoint-guided recovery mechanism. A GRU aggregates frame summaries and

temporal differences, while gated residual injection enhances the current visual representation. A joint DPO and behavior cloning objective learns from matched successful and failed trajectories. Runtime monitoring detects persistent anomalies and triggers bounded corrective control. Across ten LIBERO Spatial tasks and three random seeds, the mean success rate increased from 80.7% to 82.4%, with Task10 improving from 56.7% to 90.0%. Peak GPU memory increased by only 0.109 GiB.

However, several limitations remain, TPRVLA still exhibits limited decision making performance on some long horizon manipulation tasks, particularly when the target object is positioned close to surrounding objects. Its ability to distinguish closely arranged objects and select appropriate actions under these conditions requires further improvement.

Future work will explore adaptive temporal memory, more reliable preference construction and flexible recovery controllers for diverse failure conditions. Broader evaluation across manipulation benchmarks, environmental disturbances and real robot platforms will assess generalization and practical deployment. These efforts will aim to achieve more consistent task improvements while reducing execution overhead.

Data availability

The publicly available LIBERO dataset used in this study can be accessed at https://github.com/Lifelong-Robot-Learning/LIBERO.

References

[1] Brohan, A. *et al.* Do as i can, not as i say: Grounding language in robotic affordances. *Conference on robot learning*, 287–318 (Pmlr, 2023).

[2] Driess, D. *et al.* Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378* (2023).

[3] Brohan, A. *et al.* Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817* (2022).

[4] Brohan, A. *et al.* Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818* (2023).

[5] O'Neill, A. *et al.* Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 6892–6903 (IEEE, 2024).

[6] Liu, B. *et al.* Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems* **36**, 44776–44791 (2023).

[7] Khazatsky, A. *et al.* Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945* (2024).

[8] Chi, C. *et al.* Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *Proceedings of Robotics: Science and Systems* (Daegu, Republic of Korea, 2023).

[9] Zhao, T. Z., Kumar, V., Levine, S. & Finn, C. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. *Proceedings of Robotics: Science and Systems* (Daegu, Republic of Korea, 2023).

[10] Agia, C. *et al.* Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress. (eds Agrawal, P., Kroemer, O. & Burgard, W.) *Proceedings of The 8th Conference on Robot Learning*, Vol. 270 of *Proceedings of Machine Learning Research*, 689–723 (PMLR, 2025).

[11] Huang, W. *et al.* Inner monologue: Embodied reasoning through planning with language models. (eds Liu, K., Kulic, D. & Ichnowski, J.) *Proceedings of The 6th Conference on Robot Learning*, Vol. 205 of *Proceedings of Machine Learning Research*, 1769–1782 (PMLR, 2023).

[12] Kim, M. J. *et al.* Openvla: An open-source vision-language-action model. (eds Agrawal, P., Kroemer, O. & Burgard, W.) *Proceedings of The 8th Conference on*

Robot Learning, Vol. 270 of *Proceedings of Machine Learning Research*, 2679–2713 (PMLR, 2025).

[13] Karamcheti, S. *et al.* Prismatic VLMs: Investigating the design space of visually-conditioned language models. (eds Salakhutdinov, R. *et al.*) *Proceedings of the 41st International Conference on Machine Learning*, Vol. 235 of *Proceedings of Machine Learning Research*, 23123–23144 (PMLR, 2024).

[14] Oquab, M. *et al.* DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research* (2024). Featured Certification.

[15] Zhai, X., Mustafa, B., Kolesnikov, A. & Beyer, L. Sigmoid loss for language image pre-training. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 11941–11952 (2023).

[16] Hu, E. J. *et al.* LoRA: Low-rank adaptation of large language models. *International Conference on Learning Representations* (2022).

[17] Yang, T. *et al.* Aim: Adapting image models for efficient video action recognition. *arXiv preprint arXiv:2302.03024* (2023).

[18] Shi, H. *et al.* Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *International Conference on Learning Representations*, Vol. 2026, 18567–18602 (2026).

[19] Houlsby, N. *et al.* Parameter-efficient transfer learning for NLP. (eds Chaudhuri, K. & Salakhutdinov, R.) *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97 of *Proceedings of Machine Learning Research*, 2790–2799 (PMLR, 2019).

[20] Rafailov, R. *et al.* Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems* **36**, 53728–53741 (2023).

[21] Liu, Z., Bahety, A. & Song, S. Reflect: Summarizing robot experiences for failure explanation and correction. (eds Tan, J., Toussaint, M. & Darvish, K.) *Proceedings of The 7th Conference on Robot Learning*, Vol. 229 of *Proceedings of Machine Learning Research*, 3468–3484 (PMLR, 2023).

[22] Zhao, G. *et al.* Flare: A failure-aware framework for autonomous correction and recovery in visual-language robotic manipulation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 22391–22401 (2026).

[23] Yao, S. *et al.* React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations (ICLR)* (2023).

[24] Mees, O. *et al.* Octo: An open-source generalist robot policy. *First workshop on vision-language models for navigation and manipulation at ICRA 2024* (2024).

[25] Black, K. *et al.* π_0: A Vision-Language-Action Flow Model for General Robot Control. *Proceedings of Robotics: Science and Systems* (LosAngeles, CA, USA, 2025).

[26] Liu, S. *et al.* Rdt-1b: a diffusion foundation model for bimanual manipulation. *International Conference on Learning Representations*, Vol. 2025, 29982–30009 (2025).

[27] Li, X. *et al.* Vision-language foundation models as effective robot imitators. *International Conference on Learning Representations*, Vol. 2024, 26703–26721 (2024).

[28] Jiang, Y. *et al.* VIMA: Robot manipulation with multimodal prompts. (eds Krause, A. *et al.*) *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202 of *Proceedings of Machine Learning Research*, 14975–15022 (PMLR, 2023).

[29] Belkhale, S. *et al.* RT-H: Action Hierarchies using Language. *Proceedings of Robotics: Science and Systems* (Delft, Netherlands, 2024).

[30] Hori, K., Suzuki, K. & Ogata, T. Enhancement of long-horizon task planning via active and passive modification in large language models. *Sci. Rep.* **15**, 7113 (2025).

[31] Hentosh, L., Hladun, Y., Tsmots, I. & Bisikalo, O. Depth-augmented diffusion policy with pseudo-depth for robust robotic manipulation. *Sci. Rep.* **16**, 28233 (2026).

[32] Walke, H. R. *et al.* Bridgedata v2: A dataset for robot learning at scale. (eds Tan, J., Toussaint, M. & Darvish, K.) *Proceedings of The 7th Conference on Robot Learning*, Vol. 229 of *Proceedings of Machine Learning Research*, 1723–1736 (PMLR, 2023).

[33] Nair, S., Rajeswaran, A., Kumar, V., Finn, C. & Gupta, A. R3m: A universal visual representation for robot manipulation. (eds Liu, K., Kulic, D. & Ichnowski, J.) *Proceedings of The 6th Conference on Robot Learning*, Vol. 205 of *Proceedings of Machine Learning Research*, 892–909 (PMLR, 2023).

[34] Saxena, S., Sharma, M. & Kroemer, O. Mrest: Multi-resolution sensing for real-time control with vision-language models. *arXiv preprint arXiv:2401.14502* (2024).

[35] Xia, W. *et al.* Human-assisted robotic policy refinement via action preference optimization. *Advances in Neural Information Processing Systems* **38**, 36746–36768 (2026).

[36] Duan, J. *et al.* AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. *1st Workshop on X-Embodiment Robot Learning* (2024).

[37] Wu, Y., Tian, T., Swamy, G. & Bajcsy, A. From foresight to forethought: VLM-in-the-loop policy steering via latent alignment. *Second Workshop on Out-of-Distribution Generalization in Robotics at RSS 2025* (2025).

Author contributions

Sun, Shi, and Chu jointly designed the experiments. Sun led the experimental work and manuscript writing. Han and Liu contributed to portions of the manuscript and to data processing. All authors reviewed the manuscript.

Funding

This work was supported by Undergraduate Training Programs for Innovations by NEFU [grant number S202610225258].

Additional information

Competing interests

The authors declare no competing interests.

Figure legends

Figure 1. Illustration of a robotic manipulation task guided by a language instruction.

Figure 2. Overall Framework of TPRVLA.

Figure 3. Initial simulation states of the ten evaluation tasks in the LIBERO Spatial benchmark. Panels (a)–(j) correspond to Tasks 1–10, respectively.

Figure 4. The comparison of task execution under the same instruction and initial condition. The top row shows that OpenVLA abandons the task after a failed grasp attempt. The bottom row shows that TPRVLA detects the failure, retries the grasp, and successfully completes the task through its closed-loop recovery mechanism.

Figure 5. Ablation results on four representative LIBERO Spatial tasks. Each task is evaluated over 30 rollouts using random seed 7. The mean denotes the average success rate across the four tasks.

Figure 6. Sensitivity of the Task7 success rate to the Temporal Adapter injection strength. Each result is calculated from the first 10 evaluation rollouts under random seed 7 with $t = 5$, while all other experimental settings remain fixed.

Figure 7. Sensitivity of the Task10 success rate to the Temporal Adapter learning rate. Each result is calculated from the first 10 evaluation rollouts under random seed 7 while all other experimental settings remain unchanged.

Figure 8. Sensitivity of the Task10 success rate to the number of DPO training steps. Each result is calculated from the first 10 evaluation rollouts under random seed 7 while all other experimental settings remain unchanged.

Figure 9. Model efficiency comparison between OpenVLA and TPRVLA on a single NVIDIA GeForce RTX 4090 GPU. Panels (a) and (b) report the average single step inference time and average rollout time on Task1, Task7, and Task10, together with their overall results. Panel (c) reports the peak GPU memory consumption. TPRVLA increases the peak GPU memory by only 0.109 GiB, while increasing the average single step inference time and average rollout time by 12.4% and 13.1%, respectively.

Tables

Table 1 The ten manipulation tasks in LIBERO-Spatial.

Task	Language Instruction
1	Pick up the black bowl between the plate and the ramekin and place it on the plate.
2	Pick up the black bowl next to the ramekin and place it on the plate.
3	Pick up the black bowl from the table center and place it on the plate.
4	Pick up the black bowl on the cookie box and place it on the plate.
5	Pick up the black bowl in the top drawer of the wooden cabinet and place it on the plate.
6	Pick up the black bowl on the ramekin and place it on the plate.
7	Pick up the black bowl next to the cookie box and place it on the plate.
8	Pick up the black bowl on the stove and place it on the plate.
9	Pick up the black bowl next to the plate and place it on the plate.
10	Pick up the black bowl on the wooden cabinet and place it on the plate.

Table 2 Core training parameters of the Temporal Memory Adapter and direct preference optimization

Module	Parameter	Setting
Temporal Adapter	Trainable parameters	Approximately 55.6 M
	Temporal window size	$T = 5$
	Memory dimension	1024
	Residual scale	0.1
	Optimizer and learning rate	AdamW, 5×10^{-5}
	Batch size and accumulation	1, 1
	Training steps	1000
DPO Adapter	LoRA parameters	$r = 8$, $\alpha = 8$
	Optimizer and learning rate	AdamW, 5×10^{-6}
	Batch size and accumulation	1, 8
	DPO coefficient β	0.2
	BC weight	0.03
	Label smoothing	0.05
	Training steps and checkpoint	100, Step 80

Table 3 Task success rate (%) comparison of different methods on LIBERO Spatial.

Method / Setting	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Mean
BC-Transformer	83.3	93.3	70.0	80.0	56.7	33.3	90.0	66.7	70.0	36.7	68.0
Diffusion Policy	86.7	96.7	80.0	86.7	63.3	43.3	93.3	76.7	80.0	60.0	76.7
Octo	86.7	**100.0**	80.0	90.0	66.7	43.3	96.7	80.0	83.3	53.3	78.0
OpenVLA	90.0	**100.0**	**90.0**	**96.7**	**73.3**	**50.0**	93.3	76.7	80.0	56.7	80.7
TPRVLA (Seed 7)	93.3	93.3	80.0	90.0	73.3	40.0	100.0	86.7	83.3	90.0	83.0
TPRVLA (Seed 17)	93.3	93.3	80.0	93.3	63.3	40.0	100.0	83.3	86.7	90.0	82.3
TPRVLA (Seed 36)	93.3	93.3	80.0	93.3	63.3	40.0	100.0	83.3	83.3	90.0	82.0
TPRVLA (Mean)	**93.3**	93.3	80.0	92.2	66.7	40.0	**100.0**	**84.4**	**84.4**	**90.0**	**82.4**

T1–T10 denote Task1–Task10, respectively. For TPRVLA, each seed was evaluated over 30 rollouts per task, and the reported mean was calculated from 90 rollouts across Seeds 7, 17, and 36.

Table 4 Runtime Anomaly Recovery Statistics for Task1, Task3, Task7 and Task8

Task	Evaluation Episodes	Intervention Episodes	Recovery Attempts	Successful Recovery Episodes	Success After Recovery (%)
Task1	30	3	3	2	66.7
Task3	30	4	5	2	50.0
Task7	30	1	1	1	100.0
Task8	30	12	15	9	75.0
Total	**120**	**20**	**24**	**14**	**70.0**

Note: Each task is evaluated over 30 episodes using random seed 7 and initial state indices 0 to 29. A successful recovery episode contains an actual intervention and subsequently achieves task success.